# 扩展KMP算法

湖南 周戈林

## 1．内容介绍

串匹配问题是一类实用价值很高的信息学问题。对于这类问题，我们一般使用功能强大的后缀树和后缀数组加以解决。传统的KMP算法由于功能简单，往往不受重视。

但值得注意的是，后缀树和后缀数组的编程复杂度很高，程序运行常数极大，空间耗费也不小，在解决数据范围很大的问题时显得力不从心。在某些情况下，运用KMP算法的改进，也就是扩展KMP算法可以恰到好处地解决问题。

## 2．主算法

给定一个串S，定义*n*=|S|，extend[*i*]表示S与S[*i*…*n*]的最长公共前缀长度。我们可以在线性时间内得到所有的extend[*i*]。

鉴于已有论文对这个算法进行细致、感性的描述，在此就不花费篇幅赘述这一算法。如果您想详细了解这一算法，可以参见《寻找最大重复子串》（作者林希德）和《求最长回文子串与最长重复子串》（作者何林）。

## 3．算法的应用

### 例一：模板

Byteasar想用一条相当长的图案（内容是一个字符串）装饰他的屋子。为了这个，他必须准备一个适当的模板，这个模板跟图案的前一段相同。他将会把模板放在墙壁的某些位置，然后按照模板进行描绘。他每次只能描绘当时模板覆盖到的所有位置，不能只描绘其中的一些。同时只要字母符合，墙壁的每个位置都可以被描绘多次。模板上所有的字母都是相邻的，其中没有空格。当然，我们总是能找到一个模板能拼出整个图案。但是Bytesar希望模板花费最小，也就是这个模板尽量短。

**任务**

写一个程序：

从标准输入中读入Byteasar想要描绘在墙壁上的图案，

计算出完全描绘这个图案所需的模板的最短长度，

把结果写到标准输出中。

**输入描述**

标准输入仅包含一行，就是Byteasar想描绘在墙壁上的图案，它包含最多500,000个英文小写字母。

**输出描述**

标准输出仅包含一行，就是最短模板长度。

**样例**

对于以下的输入数据：

ababbababbabababbabababbababbaba

正确的输出应该是：

8

样例的结果是这么来的：

**ababbababbabababbabababbababbaba**

**ababbaba**

**ababbaba**

**ababbaba**

**ababbaba**

**ababbaba**

**解法分析**

本题要求找一个最短的模板串，把主串拼出来。那么怎么样的模板串才能拼出主串呢？

首先，这个模板串既是主串的前缀，也是主串的后缀。因为无论怎么拼，总要拼主串的第一个和最后一个字符，而且不能多拼字符，所以模板串肯定要跟主串的前一段和后一段相等。

其次，模板串要能覆盖整个主串，也就是说每次描绘到的位置的并必须是整个主串。

以上两点性质虽然十分简单，但这为解题指明了方向：第一点确定了模板串的选择范围，第二点确定了判定一个模板是否合法的方法。

我们定义主串为S,*n*=|S|，那么模板串T=S[1…*m*],1≤*m*≤*n*，我们希望*m*最小。

显然有以下算法：从小到大依次枚举模板串的长度*m*，求出T=S[1…*m*]在S中所有能匹配到的位置，从左到右依次记为*p*1, *p*2 ,…,*pk* ，显然*p*1=1。如果对于任意的1≤*i*≤*k*-1都有*pi*+1-*pi*≤*m*，那么这就是一个合法的模板串，输出它的长度再退出即可。这个算法很简洁，唯一但是致命的缺点就是复杂度太高。即使用效率很高的KMP算法来寻找匹配位置，对于每个被枚举的*m*也需要O(*n*)的时间来判断是否可行。总复杂度高达O(*n*2)，完全不具有可行性。

我们研究一下朴素算法的细节，考察一下模板串的长度变大会导致什么变化。

**[定理1]**当*m*从*m*0变为*m*0+1时，某些匹配点可能会消失，但是绝对不会出现新的匹配点。

**[证明]**因为命题的前一部分有“可能”这个词，所以我们只需要证明不会出现新的匹配点。而这一点的证明十分简单：如果新出现了某个匹配点*q*，那么就有S[1…*m*0+1]=S[*q*…*q*+*m*0]，显然也有S[1…*m*0]=S[*q*…*q*+*m*0-1]，也就说*q*在以前就是一个匹配点，与假设矛盾。

定理1说明随着*m*的增大，的值是单调不减的，而*m*的值在不断增加，如果在某一时刻有*l*≤*m*且*pk*=*n*-*m*+1就得到了最优解。现在我们的问题就是怎么在枚举*m*的过程中维护*l*。

这时就可以用到上文提到的扩展KMP算法。首先预处理算出所有的extend[*i*]。当枚举的*m*≤extend[*i*]时，*i*就是一个匹配点。开始时我们把所有的*i*，也就是*m*=0时的匹配点存储到一个双向链表中，这样删除一个匹配点和更新*l*的值只需要O(1)时间。尔后，我们从小到大枚举*m*，在枚举的同时删除不合要求的匹配点，当第一个满足要求的*m*出现时算法停止。假设当前枚举*m*到*m*0，就把extend[*i*]=*m*0-1的*i*从链表中删去，可以用哈希表辅助这一步。

预处理的复杂度是O(*n*)的，同时每个位置只需要进一次链表、出一次链表，而链表维护的费用是O(1)，因此整个算法的时间复杂度就是线性的，是一个非常优秀的算法。

### 例二：单词的周期

一个串被定义为一个有限的英文小写字母序列。特别的，这个序列可以为空，也就是说不包含任何字母。我们定义A=BC表示串A可以通过连接（把一个串接在另一个串的后面，也就是没有空格隔开）串B和串C得到（按照这种顺序）。如果存在两个串P和B把它们连接之后能得到串A，也就是A=PB，我们就称P是A的前缀。换句话说，A的前缀就是串A最前面的片断。附加的，如果P≠A并且串P不为空，我们就说P是A的严格前缀。

我们说串Q是串A的一个周期，仅当Q是串A的严格前缀，并且串A是串QQ的前缀（不一定是严格前缀）。举例来说，串abab和ababab都是串abababa的周期。串A的最长周期是它的周期中最长的一个，也有可能是空串，如果A没有任何周期。举例来说，串ababab的最长周期是abab。串abc的最长周期是空串。

**任务**

写一个程序：

从标准输入内读入一个串的长度以及这个串，

统计这个串的所有前缀的最长周期长度和，

把结果写到标准输出中。

**输入描述**

标准输入第一行是一个整数*n*（1≤*n*≤1,000,000）——串的长度。接下来一行有一个包含*n*个英文小写字母的序列——串的内容。

**输出描述**

标准输出仅包含一行，就是标准输入中串的所有前缀的最长周期长度和。

**样例**

对于以下的输入数据：

8

babababa

正确的输出应该是：

24

样例的结果是这么来的：

**b 最长周期为0**

**ba 最长周期为0**

**bab 最长周期为2**

**baba 最长周期为2**

**babab 最长周期为4**

**bababa 最长周期为4**

**bababab 最长周期为6**

**babababa 最长周期为6**

**解法分析**

把主串称作串A，且*n*=|A|。我们跳过串的周期的定义，直接定义串的最长周期，这需要借助extend函数。我们先预定义*f* [*i*]=*i*+extend[*i*+1]。

**[定理2]**若串A[1…*m*]存在周期，那么它的最长周期就是Q=A[1…*k*]，其中*k*≤*m*-1, *f* [*k*]≥*m*并且*k*的值尽量大。

**[证明]**定理2中的最长周期一定是串A的严格前缀，所以只需证明串A是串QQ的前缀，我们使用反证法证明。

如果串A不是串QQ的前缀，那么只有两种可能的情况：

1．|A|≤|QQ|且A[1…*m*]≠QQ[1…*m*]；

2．|A|＞|QQ|。

由于*f*[*k*]≥*m*，因此有Q[1…*m*-*k*]=A[*k*+1…m]，又Q=A[1…*k*]，所以有QQ[1…*m*]=A[1…*m*]。这说明情况1不可能出现。

假设情况2出现，由对情况1的分析知必有A[1..2*k*]=QQ。因为*f* [*k*]≥*m*，所以A[1..*m*-*k*]=A[*k*+1…*m*]，也就有A[1…*k*]=A[*k*+1…2*k*]=…=A[(*p*-1)*k*+1…*pk*]，此外还有A[*pk*+1…*m*]=A[1…*m*-*pk*]，其中。以上等式可用图1表示：

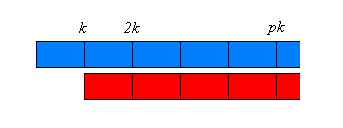


图1

由以上讨论知*f* [*pk*]≥*m*，这与*k*的最优性不符，所以情况2也不可能出现。

在否定了两种情况后，定理2的正确性就是显然的了。

直接应用定理2就可以得到以下算法：对于每个*m*，从大到小枚举对应的*k*，直到找到一个满足*f* [*k*]≥*m*的*k*后停止，把这时的*k*值累加。这个算法在预处理之后需要两重循环枚举*m*和*k*，在最坏情况下的复杂度是O(*n*2)，太高了。

注意到以下事实：假设我们从小大到大枚举*m*，恰好枚举到*m*=*m*0，又存在*i*＜*j*＜*m*0且*f* [*i*]≤*f* [*j*]，那么A[1…*i*]在这之后肯定不会成为某个A[1…*m*],*m*＞*m*0的最长周期。这是因为当*i*和*j*都是周期时*j*比*i*更优，而当*j*不是周期的时候*i*也一定不是周期。

我们可以维护一个栈*k*1, *k*2, *k*3,…, *kp*记录当前有可能成为最长周期的位置，它们满足，以及。假设我们处理到A[1…*i*]：就先判断*f* [*kp*]与*f* [*i*-1]的大小关系，如果*f* [*kp*]≤*f* [*i*-1]就将栈顶元素出栈，直到*f* [*kp*]＞*f* [*i*-1]为止，再把*i*-1入栈。此后再将判断*f* [*kp*]与*i*的大小关系，如果*f* [*kp*]＜*i*就将栈顶元素出栈，直到*f* [*kp*]≥*i*或者栈空为止。这时如果栈空说明A[1…*i*]没有周期，否则其最长周期为A[1…*kp*]。

每个位置最多进一次栈出一次栈，所以对栈的维护总共需要O(*n*)的时间。本题的时间复杂度就是O(*n*)。

## 4．总结

在本文所述的两道例题的解答过程中，我们跳出了一般的思维惯性，避开了繁琐的高级数据结构，同时尽可能地将题目条件与extend函数建立关系，从而使得可以用扩展KMP来解题。但仅仅这样是不足以得到最优算法的，我们还要对extend函数进行细致的分析，利用串的特性来解决问题，才能得到完善的解法。

## 5．附录

本文选用了两道Polish Olympiad in Informatics的题目，题目原文见附件中的sza.htm和okr.htm。

本文源程序为sza.pas和okr.pas。

本文参考了《寻找最大重复子串》（作者林希德）和《求最长回文子串与最长重复子串》（作者何林）两篇论文。